Methodes spectrales paralleles et applications aux simulations de couches de melange compressibles

Jean-Michel Male, Loula Fatima Fezoui

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

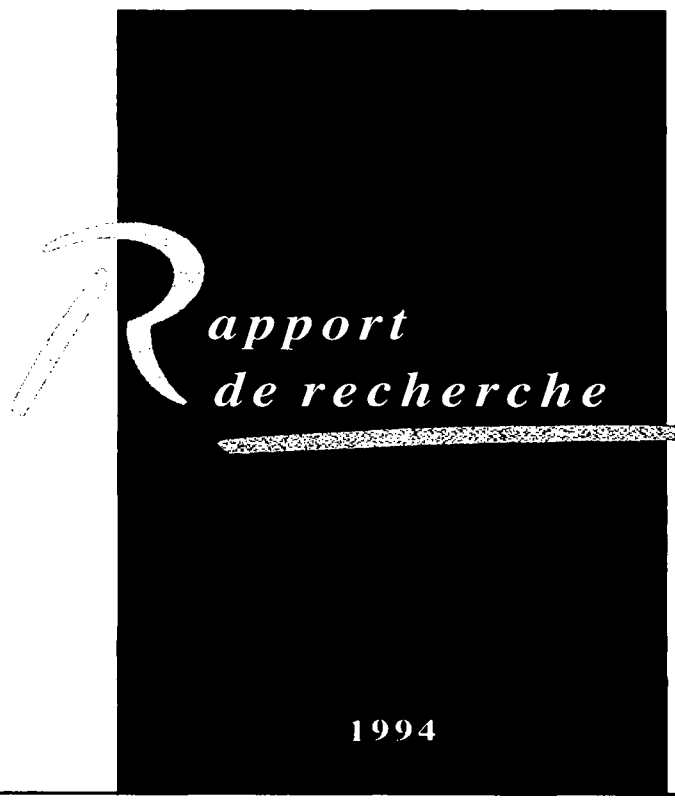
Méthodes spectrales parallèles et application aux simulations de couches de mélange compressibles

Jean-Michel MALÉ
Loula FEZOUI

N° 2107
Mars 1994

PROGRAMME 6

Calcul scientifique,
modélisation et
logiciels numériques



Méthodes Spectrales Parallèles et Application aux Simulations de Couches de Mélange compressibles

Jean–Michel MALÉ, Loula FEZOUI

Programme 6 — Calcul scientifique, modélisation et logiciel numérique
Projet Sinus

Rapport de recherche n ° 2107 — March 1994 — 38 pages

Résumé : La résolution des équations de Navier–Stokes en méthodes spectrales pour des écoulements compressibles peut être assez gourmande en temps de calcul. On étudie donc ici la parallélisation d'un tel algorithme et son implantation sur une machine massivement parallèle, la Connection–Machine CM–2. La méthode spectrale s'adapte bien aux exigences du parallélisme massif, mais l'un des outils de base de cette méthode, la Transformée de Fourier Rapide (lorsqu'elle doit être appliquée sur *les deux* dimensions d'un tableau), y est en revanche assez peu performant; d'autre part, en raison de contraintes sur le pas de temps, il est nécessaire de ne pas utiliser un trop grand nombre de points, ce qui ne permet pas de profiter de toute la puissance de la CM–2.

Mots-clé : méthodes spectrales, écoulements compressibles visqueux, simulation numérique directe, architectures parallèles.

(Abstract: pto)

Unité de recherche INRIA Sophia-Antipolis
2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex (France)
Téléphone : (33) 93 65 77 77 – Télécopie : (33) 93 65 77 65

Parallel Spectral Methods and their Application to Compressible Mixing Layers Simulations

Abstract: The resolution of Navier–Stokes equations for compressible flows using spectral methods can be CPU intensive. We study here the parallelization of such an algorithm, and its implementation on a massively parallel computer, the Connection–Machine CM–2. The spectral method is found to be well adapted to massively parrallel architectures, but one of the basic tools needed for this method, the Fast Fourier Transform, has poor efficiency on such computers when it has to be applied on the *two* dimensions of an array. Furthermore, the time integration suffers from hard restrictions if the number of points is big; this impeedes taking advantage of the whole power of the CM–2.

Key-words: spectral methods, compressible viscous flows, direct numerical simulation, parallel architectures.

1 Introduction

La parallélisation d'un algorithme de résolution des équations de la mécanique des fluides (Euler ou Navier–Stokes) se heurte à plusieurs types de difficultés, liées aux méthodes d'approximation employées. De manière générale, pour les problèmes résolus et en considérant des intégrations temporelles explicites, on peut classer ces méthodes numériques en deux grandes familles. Les éléments finis, les volumes finis et les différences finies sont des méthodes locales; ceci signifie que la discrétisation des opérateurs aux dérivées partielles en espace, en un point (ou sur une cellule), ne fera intervenir que les voisins topologiques de ce point ou de cette cellule. Dans la deuxième catégorie, on classera les méthodes spectrales et pseudo–spectrales; pour celles–ci, la valeur en un point de la dérivée partielle en espace d'une fonction dépend des valeurs de la fonction sur *tous* les points dans la direction d'espace considérée. Pour obtenir une méthode locale d'ordre p, la discrétisation des dérivées spatiales doit prendre en compte les valeurs de la fonction sur les voisins d'ordre p du point, les voisins d'ordre p se définissant par récurrence comme étant les voisins (d'ordre 1) des voisins d'ordre $p-1$. Les méthodes spectrales seront donc plus précises que les méthodes locales puisque leur ordre de précision est au moins égal au nombre de points de la discrétisation; elles seront en revanche beaucoup plus coûteuses en général en nombre d'opérations que les méthodes dites locales.

La majeure partie de ces opérations est constituée de transformées de Fourier et de relations de récurrence intervenant dans le calcul de dérivées. L'algorithme résultant est aisément vectorisable comme le montrent les résultats obtenus sur des ordinateurs vectoriels comme le CRAY et le Convex C–2 [10].

Nous voulons étudier ici le comportement de ce type d'algorithme sur une architecture massivement parallèle comme la Connection Machine CM–2. Les travaux réalisés sur ce sujet portent en général sur des constructions d'algorithmes de la transformée de Fourier rapide plus adaptés au parallélisme de la CM–2. De hautes performances ont été prévues [12] et observées [13].

Les performances se dégradent–elles et dans quelle proportion lorsqu'on fait appel à la transformée de Fourier rapide dans un algorithme contenant d'autres types d'opérations, comme la multiplication de matrices par exemple ?

Travaillant sur des écoulements incompressibles, une adaptation astucieuse de la méthode spectrale qui donne de bonnes performances sur la CM–2 a été décrite dans [8]. La subtilité dans ce cas était de réécrire les équations en variable complexe, ce qui permet de ne faire de transformées de Fourier qu'en début et fin d'exécution, ainsi qu'aux temps où l'on désire visualiser les résultats. Cette idée ne peut malheureusement être appliquée aux écoulements compressibles que nous étudions ici.

Le plan de ce rapport est le suivant :

- Dans la première partie, on décrit le problème physique et sa modélisation mathématique.
- On détaille, dans la seconde partie, la méthode numérique choisie ainsi que l'algorithme qui en résulte dans sa version *scalaire*.
- La parallélisation de tout algorithme étant liée à l'architecture et au mode de fonctionnement de la machine parallèle choisie, on décrit brièvement, dans la troisième partie, la Connection Machine CM–2.
- Dans la quatrième partie, on donne une version parallèle de la méthode spectrale utilisée sur la Connection Machine.
- Des comparaisons de performance et des résultats numériques sont présentés dans la dernière partie.

2 Le problème physique

On s'intéresse au développement temporel de tourbillons dans une couche de mélange. La situation initiale est celle représentée sur la figure (1). L'écoulement décrit est instable en ce sens que la superposition d'une perturbation, aussi infime soit–elle, suffit à entraîner l'apparition de tourbillons qui vont s'épaississant et qui peuvent éventuellement s'apparier.

S'intéresser au comportement temporel de l'instabilité, c'est choisir de se déplacer à la vitesse des tourbillons, en fixant une fenêtre de l'écoulement; on suppose en outre que la croissance de l'instabilité entre l'entrée et la sortie de la fenêtre est négligeable, ce qui conduit à supposer l'écoulement périodique.

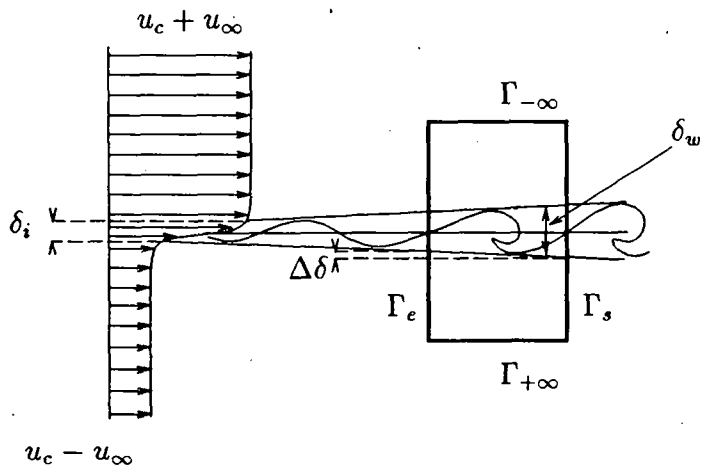


Figure 1 : Description de la couche de mélange

L'instabilité, dite de Kelvin–Helmholtz, a d'abord été étudiée pour des fluides incompressibles non visqueux [18], et visqueux [1]. Ces premiers travaux montrent l'absence de nombre de Reynolds critique; à l'inverse de la couche limite par exemple, l'instabilité n'est pas due à la viscosité, qui en atténue au contraire le développement. Le passage aux fluides compressibles a été plus délicat. Plusieurs études [2, 3, 7], toutes menées dans le cas non visqueux, ont permis de démontrer que la Compressibilité, si elle diminuait le taux d'amplification des perturbations, n'empêchait pas l'instabilité de se développer. Il faut attendre [4] et [19] pour avoir un paramètre permettant de quantifier convenablement la compressibilité de l'écoulement : le nombre de Mach convectif. La viscosité a le même effet en compressible qu'en incompressible. D'autres phénomènes intéressants ont été mis en évidence par des études théoriques ou numériques; par exemple, lorsque le nombre de Mach convectif dépasse 0.6, les perturbations tridimensionnelles se développent plus vite que leurs homologues bidimensionnelles [20]. Pour un nombre de Mach convectif de 0.8, des chocs apparaissent dans les simulations bidimensionnelles, mais n'ont pas encore été observés dans les simulations tridimensionnelles.

Nous nous limitons ici aux nombres de Mach convectifs inférieurs à 0.6. Ceci nous permet de n'envisager que des perturbations bidimensionnelles et donc de supposer le problème lui-même bidimensionnel. De même, nous évitons la présence d'ondes de choc que les méthodes spectrales capturent

difficilement. La méthode numérique développée permet cependant de traiter le cas de nombres de Mach convectifs jusqu'à 0.8, avant l'apparition de chocs [10, 9].

2.1 Equations

L'écoulement est instationnaire; il est modélisé par les équations de Navier-Stokes. On suppose que le fluide est un gaz parfait dont les chaleurs spécifiques sont constantes. La viscosité et la conductivité thermique dépendent de la température suivant la loi de Sutherland. Les équations sont adimensionnées en choisissant l'épaisseur initiale de vorticité δ_i, la différence des vitesses supérieure et inférieure, la température et la masse volumique initiales sur la frontière supérieure du domaine comme grandeurs de référence. Le temps de référence est le quotient de l'épaisseur initiale de vorticité par la différence des vitesses supérieure et inférieure; la viscosité et la conductivité thermique sont adimensionnées par leurs valeurs sur la frontière supérieure du domaine. Les équations adimensionnées s'écrivent :

$$\left\{\begin{array}{rcl} \rho_t + (\rho u)_x + (\rho v)_y & = & 0 \\ (\rho u)_t + (\rho u^2 + p)_x + (\rho uv)_y & = & \frac{1}{Re}\left[(\tau_{11})_x + (\tau_{12})_y\right] \\ (\rho v)_t + (\rho uv)_x + (\rho v^2 + p)_y & = & \frac{1}{Re}\left[(\tau_{12})_x + (\tau_{22})_y\right] \\ p_t + \gamma p(u_x + v_y) + up_x + vp_y & = & \frac{1}{ReM_*^2 Pr}\left[(\lambda T_x)_x + (\lambda T_y)_y\right] \\ & + & \frac{\gamma - 1}{Re}\left[\tau_{11}u_x + \tau_{12}(u_y + v_x) + \tau_{22}u_y\right] \end{array}\right. \quad (1)$$

où $\tau_{11} = \frac{2\mu}{3}(2u_x - v_y)$, $\tau_{12} = \mu(u_y + v_x)$ et $\tau_{22} = \frac{2\mu}{3}(2v_y - u_x)$ sont les composantes du tenseur des taux de déformation.

Les nombres sans dimension qui apparaissent dans ces équations sont les nombres de Reynolds Re, de Mach M_* et de Prandtl Pr :

$$Re = \frac{2u_\infty \delta_i \rho_\infty}{\mu_\infty}\ ,\ M_* = \frac{2u_\infty}{\sqrt{\gamma \mathcal{R} T_\infty}} = 2M_c\ ,\ Pr = \frac{C_p \mu_\infty}{k_\infty}$$

M_c est le nombre de Mach convectif et l'indice $_\infty$ indique qu'il s'agit des valeurs sur le bord supérieur du domaine de calcul. La loi d'état adimensionnée s'écrit :

$$p = \frac{1}{\gamma M_*^2}\rho T$$

La loi de Sutherland adimensionnée a pour expression :

$$\mu = T^{3/2}\frac{1+\frac{T_1}{T_\infty}}{T+\frac{T_1}{T_\infty}}$$

T_1 est une constante homogène à une température.

On résout d'autre part une équation d'advection–diffusion sous forme conservative; l'inconnue y est un scalaire passif, noté Y, et l'équation adimensionnée s'écrit :

$$(\rho Y)_t + (\rho u Y)_x + (\rho v Y)_y = \frac{1}{Re\, Sc}(Y_{xx} + Y_{yy}) \qquad (2)$$

Cette équation traduit simplement la conservation de la quantité (ρY). Si maintenant on suppose d'abord que le fluide est composé de deux espèces chimiques de caractéristiques physiques identiques ne réagissant pas entre elles, qu'ensuite les coefficients de diffusion moléculaire D_k sont égaux à D pour les deux espèces ($k = 1, 2$) et que le produit ρD est constant en espace et en temps, enfin que le terme de couplage qui apparaît dans l'équation de pression est nul, alors on peut identifier Y avec la fraction massique de l'une des espèces. Le nombre de Schmidt qui apparaît dans (2) est alors défini par :

$$Sc = \frac{\mu}{\rho D} = LePr$$

Le scalaire Y est dit passif parce qu'il n'influence pas l'écoulement. L'équation (2) peut–être vue comme celle d'un marqueur caractérisant l'écoulement.

2.2 Conditions initiales

L'écoulement, à l'instant initial, est formé d'un état de base auquel est superposé une petite perturbation sinusoïdale. La vitesse de base présente un profil en tangente hyperbolique, la pression est constante et la température vérifie l'équation d'énergie de la couche limite avec $Pr = 1$:

$$\begin{aligned} u &= \frac{1}{2}\tanh(2y)\ ,\ v=0\ ,\ p=\frac{1}{\gamma M_*^2}=\frac{1}{4\gamma M_c^2} \\ T &= 1+\frac{\gamma-1}{2}M_c^2\left(1-(2u)^2\right) \end{aligned} \tag{3}$$

La concentration initiale est :

$$Y=\frac{1}{2}(1+\tanh(2y))$$

La perturbation sinusoïdale est à divergence nulle. Elle est confinée au voisinage de l'axe $y=0$ au moyen d'une gaussienne; elle s'écrit :

$$\left\{\begin{array}{rcl} u' &=& \dfrac{\varepsilon y\lambda}{20\pi}\sin(\frac{2\pi x}{\lambda})\exp(-\frac{1}{10}y^2) \\ v' &=& \dfrac{\varepsilon}{2}\cos(\frac{2\pi x}{\lambda})\exp(-\frac{1}{10}y^2) \end{array}\right. \tag{4}$$

où ε est un nombre positif petit.

2.3 Conditions aux limites

On suppose que l'écoulement est confiné verticalement et on applique des conditions de glissement, de flux de chaleur et de pression nuls sur les frontières horizontales du domaine, notées $\Gamma_{\pm\infty}$ sur la figure (1). Si l'on continue l'analogie entre scalaire passif et concentration, la condition imposée sur cette quantité correspond à une paroi imperméable. Ces conditions aux limites s'écrivent :

$$\frac{\partial u}{\partial y}=0\ ,\ v=0\ ,\ \frac{\partial p}{\partial y}=0\ ,\ \frac{\partial T}{\partial y}=0\ ,\ \frac{\partial Y}{\partial y}=0 \tag{5}$$

Comme nous nous intéressons au développement temporel de la couche de mélange, nous imposons des conditions de périodicité en Γ_e et en Γ_s sur la figure (1). Avec $\Phi=(\rho,u,v,p,Y)^t$ et en notant Λ la période spatiale, les conditions s'écrivent :

$$\Phi(\Lambda,t)=\Phi(0,t)\ ,\ \frac{\partial\Phi}{\partial x}(\Lambda,t)=\frac{\partial\Phi}{\partial x}(0,t) \tag{6}$$

3 La méthode numérique

Rappelons brièvement les principales caractéristiques de l'approximation utilisée ici. Soit à résoudre une équation aux dérivées partielles

$$\frac{\partial u}{\partial t} - \frac{\partial}{\partial x} F(u) = 0 \tag{7}$$

u étant une fonction du temps et de la position, F étant une fonction de u suffisamment régulière.

L'approximation de la dérivée temporelle sera décrite plus tard. Il suffit simplement de savoir que le schéma en temps sera complètement explicite; on supposera provisoirement qu'il s'agit d'un schéma d'ordre un :

$$\frac{u^{n+1} - u^n}{\Delta t} = \frac{\partial}{\partial x} F(u^n) \tag{8}$$

u^{n+1} représente la valeur de u à l'instant $(n+1)\Delta t$. Il faut noter que, pour un schéma explicite, les conditions aux limites peuvent être calculées après que le schéma ait évalué u^{n+1} sur tous les points intérieurs.

3.1 Méthode de collocation

Pour l'approximation spatiale, on utilise une méthode pseudo–spectrale avec développement des inconnues en polynômes de Tchebychev. On renvoie à [5] pour un exposé complet des propriétés requises de l'opérateur et de la solution pour que la série tronquée en polynômes de Tchebychev converge uniformément vers la solution exacte de (7); on supposera ici que l'opérateur est suffisamment régulier pour qu'il existe une solution unique u, C^∞, pourvu que l'on fournisse des conditions initiales et des conditions aux limites appropriées. La solution exacte u de l'équation aux dérivées partielles (7) est approchée par son développement tronqué à l'ordre N en polynômes de Tchebychev

$$u(x,t) \approx u_N(x,t) = \sum_{k=0}^{N} \hat{u}_k(t) T_k(x) \quad \forall x \in [-1,1] \tag{9}$$

où les $\hat{u}_k(t)$ sont les coefficients (constants en x) du développement. Les polynômes de Tchebychev de première espèce sont définis par :

$$T_k(x) = \cos(k \arccos(x)) \quad x \in [-1,1] \text{ pour } k \in \mathbb{N} \tag{10}$$

Pour calculer u^{n+1}, outre la connaissance de u^n, il est nécessaire de pouvoir évaluer la dérivée spatiale de $F(u^n)$. Pour ce faire, on écrit d'abord $\partial F(u)/\partial x$ en fonction de u et de $\partial u/\partial x$. Il ne reste plus qu'à évaluer cette dernière quantité. Lorsqu'on a fait le choix d'approcher $u(x,t)$ par une série Tchebychev tronquée à l'ordre N, des considérations de précision imposent d'approcher $\partial u/\partial x$ par la dérivée de l'approximation (9) :

$$\frac{\partial}{\partial x} u_N(x,t) = \sum_{k=0}^{N} \hat{u}_k(t) T'_k(x) \tag{11}$$

On utilise une méthode de collocation, c'est à dire que l'on résout exactement l'équation aux dérivées partielles sur un certain nombre de points de collocation; leur choix dépend des fonctions de base choisies, et pour les polynômes de Tchebychev, on a coutume de choisir les extrema du polynôme de degré N :

$$x_j = \cos \frac{\pi j}{N} \tag{12}$$

D'après (10), on peut exprimer la valeur du $k^{\text{ième}}$ polynôme de Tchebychev sur le $j^{\text{ième}}$ point de collocation :

$$T_k(x_j) = \cos \frac{k\pi j}{N} \tag{13}$$

On sait que les polynômes de Tchebychev forment une base Hilbertienne de l'espace $L^2_\omega([-1,1])$ où $\omega(x) = (1-x^2)^{-1/2}$. On obtient une relation d'orthogonalité discrète en approchant l'intégration dans l'écriture du produit scalaire par une formule de Gauss-Lobatto,

$$\sum_{j=0}^{N} \frac{1}{c_j} T_k(x_j) T_l(x_j) = \frac{N}{2} c_k \delta_{kl} \quad k,l \in \{0,\ldots,N\} \tag{14}$$

avec $c_0 = c_N = 2$ et $c_k = 1$ si $k \in \{1,\ldots,N-1\}$.

3.2 La dérivation

3.2.1 Méthode directe

Elle consiste à exprimer $\hat{u}_k(t)$ en fonction des $u_N(x_j,t)$, pour tous les points de collocation x_j, puis à reporter le résultat obtenu dans l'expression (11). En effet, d'après (9),

$$u(x_j,t) \approx u_N(x_j,t) = \sum_{k=0}^{N} \hat{u}_k(t) T_k(x_j) \quad j \in \{0,\cdots,N\}$$

En multipliant membre à membre par $T_l(x_j)/c_j$, et en sommant sur j, on obtient :

$$\sum_{j=0}^{N} \frac{1}{c_j} u_N(x_j,t) T_l(x_j) = \sum_{k=0}^{N} \left(\hat{u}_k(t) \sum_{j=0}^{N} \frac{1}{c_j} T_k(x_j) T_l(x_j) \right)$$

La relation d'orthogonalité permet d'isoler $\hat{u}_l(t)$ et

$$\hat{u}_l(t) = \frac{2}{N c_l} \sum_{j=0}^{N} \frac{1}{c_j} u_N(x_j,t) T_l(x_j) \tag{15}$$

On reporte ensuite (15) dans (11) et on obtient :

$$\begin{aligned} \frac{\partial}{\partial x} u_N(x_p,t) &= \frac{2}{N} \sum_{k=0}^{N} \frac{1}{c_k} \sum_{j=0}^{N} u_N(x_j,t) T_k'(x_p) T_k(x_j) \\ &= \frac{2}{N} \sum_{j=0}^{N} u_N(x_j,t) \sum_{k=0}^{N} \frac{1}{c_k} T_k'(x_p) T_k(x_j) \\ &= \sum_{j=0}^{N} d_{jp}^{(1)} u_N(x_j,t) \end{aligned} \tag{16}$$

Les coefficients $d_{jp}^{(1)}$ sont explicités en annexe. La dérivation se réduit donc à la multiplication de la matrice $[d_{jp}^{(1)}]$ par le vecteur $[u_N(x_j,t)]$.

3.2.2 Transformées de Fourier rapides

La deuxième façon de procéder pour calculer la dérivée de U_N par rapport à x découle de la définition par récurrence des polynômes de Tchebychev :

$$2\,T_k(x) = \frac{1}{k+1}T'_{k+1}(x) - \frac{1}{k-1}T'_{k-1}(x), \quad k \geq 1 \tag{17}$$

De cette expression, on peut aussi déduire la relation de récurrence qui lie les coefficients de la fonction et ceux de la dérivée :

$$2k\hat{u}_k = c_{k-1}\hat{u}'_{k-1} - \hat{u}'_{k+1} \tag{18}$$

Il reste maintenant à exprimer les coefficients de l'approximation à l'aide des valeurs de la fonction sur les points de collocation. On écrit donc la relation d'orthogonalité discrète (14), et en notant $\underline{i}^2 = -1$, on obtient :

$$\begin{aligned}
c_k\hat{u}_k &= \frac{2}{N}\sum_{j=0}^{N}\frac{1}{c_j}u_N(x_j)\cos\frac{k\pi j}{N} \qquad \text{for } 0 \leq k \leq N \\
&= \frac{1}{N}\sum_{j=0}^{N}\frac{1}{c_j}u_N(x_j)\left(\exp(\underline{i}\frac{k\pi j}{N}) + \exp(-\underline{i}\frac{k\pi j}{N})\right) \\
&= \frac{1}{N}\left[\sum_{j=0}^{N}\frac{1}{c_j}u_N(x_j)\exp(-\underline{i}\frac{k\pi j}{N}) + \sum_{j=-N}^{-1}\frac{1}{c_{-j}}u_{-j}\exp(\underline{i}\frac{k\pi j}{N})\right] + \frac{1}{c_0}u_0
\end{aligned}$$

on effectue un changement d'indice pour arriver au résultat final :

$$\begin{aligned}
c_k\hat{u}_k &= \frac{1}{N}\left[\sum_{j=0}^{N}u_N(x_j)\exp(-\underline{i}\frac{2\pi kj}{2N}) + \sum_{j=N+1}^{2N-1}\tilde{u}_N(x_j)\exp(-\underline{i}\frac{2\pi kj}{2N})\right] \\
&= \frac{1}{N}\left[\sum_{j=0}^{2N-1}\tilde{u}_N(x_j)\exp(-\underline{i}\frac{2\pi kj}{2N})\right]
\end{aligned} \tag{19}$$

où $\tilde{u}_N(x_j) = u_N(x_j)$ pour $0 \leq j \leq N$ et $\tilde{u}_N(x_j) = u_N(x2\,N - j)$ pour $N+1 \leq j \leq 2N-1$.

Le calcul de $\hat{u}_k$ se réduit ainsi à une transformée de Fourier de longueur $2N$. Le calcul de la dérivée se compose donc d'une transformée de Fourier

qui permet de calculer les coefficients $\hat{u}_k$, d'une relation de récurrence qui donne les coefficients de la dérivée $\hat{u}'_k$ et d'une transformée de Fourier inverse qui permet de récupérer les valeurs de la dérivée de l'approximation sur les points de collocation.

3.2.3 Approximation temporelle

On utilise un schéma de Runge–Kutta, précis au troisième ordre en temps, à faible stockage [26]. Le critère de stabilité nécessaire pour assurer la stabilité du schéma implique que le pas de temps soit de l'ordre de $1/N^4$, où N est le degré de l'approximation dans l'une des directions d'espace.

3.3 Algorithme

Début du programme
Initialisation de la géométrie, des TFR et de l'écoulement.
Tant que [Temps $< T_{max}$]
Faire Boucle en temps
Calcul du pas de temps $\Delta t = \min_{i,j} \Delta t_{i,j}$
Pour [chacun des pas du schéma Runge-Kutta 3]
Faire
Calcul $\mu_{i,j} = T_{i,j}^{3/2} \frac{1 + \frac{T_1}{T_\infty}}{T_{i,j} + \frac{T_1}{T_\infty}}$ et $\lambda_{i,j} = \mu_{i,j}$.
Calcul des dérivées pour les flux Euler (5 en x, 5 en y).
Assemblage point par point des flux Euler.
Calcul point par point du tenseur des contraintes τ_{ij}.
Calcul des dérivées pour les flux visqueux.
Assemblage des flux diffusifs et mise à jour de la solution pour les variables hydrodynamiques.
Conditions aux limites
Calcul des dérivées pour l'équation de concentration (3 en x, 3 en y).
Assemblage des flux et mise à jour de la concentration.
Conditions aux limites pour la concentration.
Fin
Si [Temps = Temps de stockage] **Alors**
Faire

Stockage de la solution
Fin
Fin Boucle en temps
Fin du programme

4 Ordinateurs à architecture parallèle

4.1 La Connection Machine CM–2

La CM–2 se présente comme un système informatique homogène, tant du point de vue du matériel que du logiciel.

La figure (2) ci–dessous représente une vue schématique des trois principaux composants de la CM–2 :

Un ordinateur frontal chargé du contrôle (séquentiel) du déroulement des programmes, et qui fournit en outre les outils nécessaires au développement (langages, librairies, débogueur).

Une unité de calcul parallèle contenant depuis $4k$ jusqu'à $64k$ processeurs élémentaires (ou éléments de calcul), disposant chacun d'une mémoire locale allant de $64k$-bits à $1M$-bits ($1k = 1024$ et $1M = 1024^2$).

Un système d'entrées sorties comprenant une unité de stockage sur disque parallèle, et un écran graphique adressable directement par un des bus parallèles du micro–contrôleur.

Le micro–contrôleur est le relais de l'ordinateur frontal dans l'unité de calcul. C'est à travers lui que le frontal accède (en écriture et en lecture) aux mémoires locales; le frontal lui envoie aussi le code compilé (macro–instructions); le micro–contrôleur convertit ce code en nano–instructions directement exécutables par les processeurs élémentaires; il est lui même programmé à un niveau intermédiaire, en micro–instructions. Le programmeur agit au niveau du langage structuré, mais il lui est possible de programmer en macro–instructions, ou même directement en micro–instructions.

Lorsque la taille du problème à résoudre est telle que le nombre de processeurs physiquement présents dans la machine n'est pas suffisant, celle–ci bascule automatiquement en mode virtuel, pourvu que suffisamment de mémoire soit disponible; chaque processeur simule alors plusieurs processeurs virtuels de manière séquentielle.

Lorsque l'exécution d'une tâche doit être conditionnelle, le micro–contrôleur "masque" ces instructions. A chaque processeur virtuel est affecté quatre bits d'état; l'un d'entre eux, dit de "contexte", est activé par le micro–contrôleur lors de l'exécution d'une macro–instruction conditionnelle. Tous les processeurs effectuent inconditionnellement la séquence conditionnelle, mais ne la mémorisent que ceux dont le bit de contexte a été activé à 1.

L'architecture des éléments de calcul ne sera pas détaillée ici. Il suffit de retenir qu'ils sont capables d'effectuer des calculs arithmétiques simples; ils peuvent être programmés pour effectuer des calculs en flottant avec une précision arbitraire, mais au prix d'une grande inefficacité. Ils adressent leur mémoire locale bit à bit, et cette caractéristique atteste que la Connection Machine n'était pas destinée a priori au calcul scientifique, qui ne nécessite pas de telles manipulations, mais plutôt au traitement d'images.

Les processeurs sont organisés physiquement en cartes de 16, les CM–chips; sur une carte, tous les processeurs sont physiquement interconnectés. Deux CM–chips partagent un accélérateur de calcul flottant, ainsi qu'un décodeur d'adressage, car les registres de l'accélérateur de calcul ne sont pas adressés bit à bit. Cette dernière caractéristique pénalise le temps de calcul, mais, en organisant différemment le stockage des données sur la mémoire locale, on peut s'affranchir de cette difficulté. Cela relève toutefois de la nano–instruction, et ces subtilités informatiques devraient être du ressort exclusif des compilateurs. Le premier d'entre eux qui intègre cette optimisation est la dernière version du CM–Fortran.

4.2 Les communications

Les communications sont organisées autour de deux réseaux physiques distincts :

le réseau physique du CM–chip qui connecte directement tous les processeurs élémentaires.

l'hypercube , dont les CM–chips sont les sommets.

Tous les CM–chips possèdent un "routeur" de messages qui agit comme un aiguillage; ces routeurs sont donc eux aussi sur les sommets de l'hypercube. Un message emprunte la connexion entre deux routeurs. Si le second est la destination effective du message, celui–ci est écrit, sinon, il est aiguillé sur l'arête de l'hypercube correspondant au chemin le plus court vers sa destination effective.

Cependant, un mécanisme de communications particulier a été implanté qui assure une meilleure performance des communications quand le taux de processeurs virtuels est important. En effet, lorsqu'un processeur virtuel veut échanger des informations avec ses plus proches voisins (Nord, Est, Sud et Ouest), les communications sont constituées, par importance décroissante, d'échanges entre mémoires d'un même processeur physique, puis d'échanges entre mémoires de processeurs physiques situés sur un même CM–chip, et, de façon assez marginale, par des échanges nécessitant l'utilisation des arêtes physiques de l'hypercube. On renvoie à [16] pour des évaluations précises des performances des communications en grille.

4.3 Les Langages de programmation

Comme indiqué précédemment, plusieurs niveaux de programmation sont accessibles. Les langages évolués sont au nombre de trois, CM–Fortran, C * et * Lisp. Le second est une extension du langage C habituel incluant des concepts propres au parallélisme; on en trouvera une présentation détaillée dans [22]. On s'intéressera plus particulièrement ici au CM–Fortran; cette extension de Fortran 77 est en fait pratiquement un nouveau langage.

Le jeu de macro–instructions (PARIS, [21]) constitue la base des langages ci–dessus. Leurs compilateurs acceptent des appels directs à des routines PARIS, ou des consultations de variables PARIS. On renvoie à [15] pour des exemples d'utilisation de langage structuré avec des appels à PARIS.

Le niveau de la nano–instruction est accessible, certes, mais son exploration sort largement du champ de cette étude.

4.4 Le CM–Fortran

le CM–Fortran répond au besoin d'un langage évolué destiné au calcul numérique sur la Connection–Machine. Sa base est le Fortran 77 auquel certaines fonctionnalités prévues dans la norme 90 ont été rajoutées, essentiellement des structures de contrôle et des instructions de manipulation de tableaux [24]. Ces dernières sont particulièrement adaptées au traitement des tableaux dont chaque élément réside dans la mémoire locale d'un processeur virtuel, que nous appellerons tableaux parallèles. On découvre donc :

une sémantique étendue permettant aux opérateurs et aux fonctions intrinsèques héritées de Fortran 77 de manipuler indifféremment des arguments scalaires ou des tableaux.

des sous–tableaux et des indices tableaux ainsi que de nouvelles règles d'affectation.

L'instruction `WHERE` permettant de sélectionner des éléments d'un tableau suivant la valeur de ces éléments.

des déclarations de types par attributs, se rapprochant de la syntaxe du langage Pascal.

D'autres fonctionnalités inclues dans CM–Fortran sont par exemple les structures de contrôles `CASE`, `DO WHILE`, `DO TIMES`, `ENDDO`.

L'utilisateur n'a normalement pas besoin de décider lui–même de la manière dont sera stocké un tableau en mémoire. Si ce tableau est détecté dans une instruction parallèle, il sera automatiquement stocké sur la Connection–Machine, dans les mémoires locales des processeurs virtuels; dans le cas contraire, il sera stocké sur le frontal. Une source d'erreurs importante est par conséquent d'avoir un tableau qui n'est que l'objet d'affectations "sérielles" dans un module, et de le passer en paramètre d'appel à un autre module qui va l'utiliser de manière parallèle. A la compilation, le tableau dans le premier module sera déclaré résidant sur le frontal, tandis que dans le second, il sera identifié comme un tableau parallèle. Dans ce cas, il *faut* indiquer au compilateur (par une directive `LAYOUT`) que le tableau *doit* résider sur la Connection–Machine, et ce même dans le premier module.

Enfin, CM–Fortran inclut des manipulations de tableaux tels que décalages d'éléments ou duplication N fois d'un vecteur de longueur M pour en faire une matrice $M \times N$, mettant en jeu les différents système de communications. Il n'est pas nécessaire, comme pour le langage C*, de gérer "à la main" les communications.

5 L'algorithme parallèle

5.1 Généralités

Comme le montre l'algorithme général (3.3), on peut séparer les instructions du programme en deux catégories :

les instructions ponctuelles où l'on affecte au point (i,j) le résultat d'un calcul faisant intervenir les valeurs en ce point d'autres grandeurs.

les instructions globales sont celles qui, pour l'évaluation d'une grandeur au point (i,j), dépendent des valeurs d'autres grandeurs aux points (i,k), $k \in \{0,\cdots,N_y\}$ ou aux points (k,j), $k \in \{0,\cdots,N_x\}$; ce sont en particulier les dérivées.

Sur des ordinateurs scalaires ou vectoriels, l'essentiel du temps de calcul, pour des méthodes spectrales explicites, est employé à calculer des dérivées spatiales. Il apparaît que ce sera également le cas sur un ordinateur à architecture SIMD comme la connection machine.CM–2. En effet, la complexité des instructions ponctuelles est de l'ordre de l'unité,; les instructions globales ont une complexité directement reliée à la taille du problème que l'on résout. Les instructions de communications, sur ces maillages structurés, ont une complexité en $\log_2(N)$, où N est la taille du problème à résoudre. Dans le cas le plus favorable, comme on le verra plus loin, on arrive à une complexité du même ordre pour les instructions globales, à savoir les calcul de dérivées. Dans ces conditions, il est clair que le gain en temps de calcul s'obtient d'abord en minimisant le temps passé à calculer des dérivées.

5.2 Complexité des différents algorithmes mis en jeu dans la dérivation

5.2.1 Multiplication de matrices

Il existe divers algorithmes parallèles de multiplication de matrices. Le plus performant en termes de complexité (algorithme de Cannon [24]) comporte $O(\log_2 N)$ opérations, pour la multiplication de deux matrices carrées d'ordre N. Il nécessite malheureusement l'occupation de N^3 processeurs, et pour les applications prévues, ce coût mémoire est prohibitif. L'algorithme suivant, par ordre de performance décroissante, est en $O(N)$ opérations et requiert N^2 processeurs. Il est implanté dans la bibliothèque utilisateur disponible sur la CM–2. On trouvera dans [6] une étude complète des différents algorithmes de multiplication de matrices et une comparaison de leurs performances mesurées.

5.2.2 Transformée de Fourier rapide

L'algorithme de la transformée de Fourier rapide ou "Butterfly" est décrit succinctement ci-après. on trouvera dans [14, 11] une description plus complète du procédé et des subtilités algorithmiques nécessaires à sa mise en œuvre.

On a coutume d'écrire la transformée de Fourier discrète sous la forme :

$$X_j = \sum_{k=0}^{N-1} x_k e^{-\frac{2\pi i j k}{N}} = \sum_{k=0}^{N-1} x_k W_N^{jk} \tag{20}$$

N est la longueur de la transformée de Fourier; x_k et X_k désignent les éléments d'indice k des vecteurs "donnée" et "résultat", respectivement; les valeurs $W_N^{jk} = \exp(-2\pi ijk/N)$ sont dénommés "Twiddle factors" (facteurs de torsion).

Dans la suite, on s'intéressera exclusivement au cas $N = 2^n$. On décompose la sommation pour faire apparaître $N/2$:

$$X_j = \sum_{k=0}^{\frac{N}{2}-1} x_k W_N^{jk} + \sum_{k=0}^{\frac{N}{2}-1} x_{k+\frac{N}{2}} W_N^{j(k+\frac{N}{2})} \tag{21}$$

D'après la définition de W_N^{jk}, on a $W_N^{j(k+\frac{N}{2})} = (-1)^j W_N^{jk}$. De plus, si $j = 2l$, $W_N^{jk} = W_{\frac{N}{2}}^{lk}$, et si $j = 2l+1$, alors $W_N^{jk} = W_N^k W_{\frac{N}{2}}^{lk}$. On obtient alors :

$$\begin{aligned} X_{2l} &= \sum_{k=0}^{\frac{N}{2}-1} W_{\frac{N}{2}}^{lk}(x_k + x_{k+\frac{N}{2}}) \qquad \forall l \in \{0, \cdots, \tfrac{N}{2}\} \\ X_{2l+1} &= \sum_{k=0}^{\frac{N}{2}-1} W_{\frac{N}{2}}^{lk} W_N^k (x_k - x_{k+\frac{N}{2}}) \quad \forall l \in \{0, \cdots, \tfrac{N}{2}\} \end{aligned} \tag{22}$$

Le calcul d'une transformée de Fourier directe de longueur N est remplacée par le calcul des deux vecteurs $[x_k + x_{k+\frac{N}{2}}]$ et $[W_N^k(x_k - x_{k+\frac{N}{2}})]$ et de deux transformées de Fourier de longueur $N/2$. L'itération du procédé conduit au bout de $n-1 = \log_2(N/2)$ étapes, à calculer des transformées de Fourier de longueur 2.

A chaque étape, les N processeurs sont actifs, et la transformée de Fourier est terminée après $\log_2 N$ étapes. Il convient de remarquer que l'adressage mémoire des éléments du tableau en sortie est modifié; l'algorithme délivre

le résultat en ordre binaire inverse, c'est-à-dire que l'élément d'indice *abcd* en base 2 a pour indice *dcba*, toujours en base 2, après exécution de la transformée de Fourier.

5.2.3 Relations de récurrence

Si l'on développe la relation (18), on s'aperçoit qu'elle peut se mettre sous la forme :

$$c_k \hat{u}'_k = \sum_{\substack{p=k+1 \\ p+k \text{ impair}}}^{N} p\hat{u}_p \tag{23}$$

Le terme $p\hat{u}_p$ à l'intérieur de la sommation, ne dépend pas de l'indice k. En conséquence, si l'on sait, en calculant une somme, affecter simultanément les sommes partielles, on peut écrire un algorithme de complexité $O(\log_2 N)$ pour les relations de récurrence. Cet algorithme existe et est connu sous le nom générique de "parallel prefix" ou encore "scanning". Pour expliquer son principe, on suppose que l'on doit calculer la somme des 8 premiers entiers non nuls. On les dispose dans un tableau parallèle; la première étape est schématisée sur la figure (3) : on a cumulé chaque élément avec celui situé immédiatement à sa gauche.

Le premier élément conserve sa valeur initiale car il n'a pas de voisin à sa gauche. Dans la seconde étape, on cumule la nouvelle valeur de l'élément avec le voisin de gauche d'ordre 2, c'est à dire le voisin de gauche de son voisin de gauche. La troisième étape fait se cumuler la nouvelle valeur de chaque élément avec celle de son voisin de gauche d'ordre trois, quand celui-ci existe. Au bout du compte, chaque nouvel élément du tableau contient la somme partielle (jusqu'à lui-même inclus) des premiers entiers. Cet algorithme a nécessité 3 opérations parallèles, c'est à dire $\log_2 8$. Il est clair que sa complexité sera $O(\log_2 N)$, pour tout N. Notons qu'un algorithme de réduction classique ne permet de calculer que la somme globale, et pas les sommes partielles.

5.3 Choix de l'algorithme de dérivation

Si l'on fait le bilan de la section précédente, on s'aperçoit que l'algorithme de multiplication de matrices le plus performant qui soit utilisable requiert $O(N)$ opérations. En revanche, l'ensemble transformée de Fourier rapide — relation

de récurrence — transformée de Fourier rapide ne demande que $O(\log_2 N)$ opérations. C'est la première raison pour laquelle on a choisi la seconde méthode. L'autre raison est conjoncturelle : en effet, la Connection Machine utilisée ici ne dispose (pour le moment) que d'accélérateurs de calculs flottants en simple précision. Or la multiplication de matrices peut induire des erreurs d'arrondi assez importantes : la matrice de dérivation possède des termes qui sont de l'ordre de N^2. S'ils apparaissent en facteur d'un nombre petit, mais dont la représentation machine est de l'ordre de 10^{-7}, alors une valeur de $N = 100$ donnera une contribution de 10^{-3} qui peut être très exagérée.

L'algorithme comprenant deux transformées de Fourier et une relation de récurrence est par nature moins imprécis. Une seule des transformées de Fourier contient en son sein une multiplication par N (celle où n'apparaît pas le facteur d'échelle $1/N$). Pour que l'erreur d'arrondi soit sensible à ce niveau, il faut que N soit de l'ordre de l'inverse de l'erreur machine. La relation de récurrence donne lieu à une multiplication par $N-1$ et la même remarque s'applique quant à la valeur de N rendant insupportable une erreur d'arrondi. Mais de plus, il se trouve que le coefficient de Tchebychev multiplié par $N-1$ est $\hat{u}_{N-1}$, par nature petit si l'approximation est bonne. Une solution consiste à forcer à zéro les coefficients de la fonction qui sont de l'ordre de l'erreur machine, et de cette façon, on augmente beaucoup la fiabilité de la méthode [25].

5.4 Algorithme parallèle

Début de l'algorithme parallèle

Initialisation de la géométrie, des TFR et de l'écoulement.

Tant que [Temps $< T_{max}$]

Faire Boucle en temps

Calcul du pas de temps (**réduction**) $\Delta t = \min_{i,j} \Delta t_{i,j}$

Pour [chacun des pas du schéma Runge-Kutta 3]

Faire

Calcul (affectation en parallèle) $\mu_{i,j} = T_{i,j}^{3/2} \dfrac{1 + \frac{T_1}{T_\infty}}{T_{i,j} + \frac{T_1}{T_\infty}}$ et $\lambda_{i,j} = \mu_{i,j}$.

Calcul des dérivées pour les flux Euler (5 en x, 5 en y).

Assemblage en parallèle des flux Euler.

Calcul en parallèle du tenseur des contraintes τ_{ij}.
Calcul des dérivées pour les flux visqueux.
Assemblage en parallèle des flux diffusifs et mise à jour (**en parallèle**) de la solution pour les variables hydrodynamiques.
Conditions aux limites
Calcul des dérivées pour l'équation de concentration (3 en x, 3 en y).
Assemblage en parallèle des flux et mise à jour (**parallèle**) de la concentration.
Conditions aux limites pour la concentration.
Fin
Si [Temps = Temps de stockage] **Alors**
Faire
Stockage de la solution
Fin
Fin Boucle en temps
Fin de l'algorithme parallèle

6 Résultats

6.1 Optimisation

6.1.1 Transformées de Fourier

Rangement mémoire des tableaux Comme l'on résout les équations de Navier–Stokes en deux dimensions d'espace, on sera amené à calculer des dérivées partielles en x et en y. D'autre part, pour que les performances de la transformée de Fourier disponible dans la librairie utilisateur soient optimales, des directives indiquant au compilateur CM–Fortran comment ranger les tableaux sont nécessaires. Supposons que l'on ait rangé dans un tableau l'une quelconque des quantités qui nous intéresse, et nommons le `TABLEAU`; il est de type `REAL`, pour que les dérivations aient un sens, et de dimension `(0:NX, 0:NY)`, `NX, NY` représentant les nombres de points dans les deux directions, respectivement. Supposons maintenant que l'on doive effectuer une transformée de Fourier dans l'une des directions, la première par exemple. Il est conseillé par le manuel d'utilisation de la librairie scientifique de la Connection Machine CM–2 [23], d'utiliser la directive suivante :

`CMF$␣LAYOUT TABLEAU(POIDSA:SEND,POIDSB:SEND)`
"`:SEND`" est un mode de rangement mémoire qui optimise les communications entre des éléments du tableau de telle façon que deux éléments dont les indices suivant la direction des x (ou des y) ne diffèrent que d'un bit soient sur des sommets voisins de l'hypercube. "`POIDSA`" ou "`POIDSB`" sont des entiers qui chiffrent la priorité accordée à l'une ou l'autre des directions dans le rangement. Ici, comme la transformée de Fourier doit se faire dans la première direction, c'est celle–ci qui doit être avantagée en termes de communication, et pour cela, il est nécessaire de choisir "`POIDSA`" le plus grand possible, tout en gardant "`POIDSB`" égal à 1. On désignera dorénavant par "poids" le rapport de "`POIDSA`" à "`POIDSB`".

Longueur	131072	65536	1024	512	256
Nombre de TFR	1	2	125	256	512
`(1:SEND,1:SEND)`	0.19 (54)	0.20 (48)	0.15 (40)	0.13 (40)	0.14 (32)
`(10:SEND,1:SEND)`	0.19 (53)	0.20 (48)	0.15 (39)	0.10 (52)	0.09 (48)
`(100:SEND,1:SEND)`	0.20 (52)	0.21 (47)	0.14 (40)	0.10 (52)	0.05 (94.8)

Tableau 1 : Influence de la longueur du vecteur sur la performance de la transformée de Fourier (TFR), temps CPU moyen par 100 transformées de Fourier, pour plusieurs combinaisons des répartitions mémoire.

On a mesuré l'effet de ce "poids" sur la performance de la transformée de Fourier rapide. Les résultats sont reportés dans le tableau (1). Les essais ont été effectués de manière à ne pas sentir l'influence du "pipe–line" scalaire de l'accélérateur de calculs flottants. Il ressort de ce tableau que l'influence du "poids" ne se fait sentir que pour des valeurs de `NY` assez élevées, se traduisant alors par un net avantage pour les essai avec le "poids" égal à 100. Ceci est tout à fait normal dans la mesure où, lorsque `NY` est faible et se rapproche de 0 (1 seule transformée de Fourier sur un tableau monodimensionnel), on s'attend à ce que le "poids" n'ait plus d'influence sur la performance. En revanche, pour `NY` égal à 256 et 512, on note un écart important entre les résultats obtenus avec un rangement `(1:SEND,1:SEND)` pour lequel aucun des deux axes n'est privilégié, et les deux autres. La performance maximale est obtenue pour une parallélisation maximale dans la direction où il n'y a pas de transformée de Fourier.

Ces performances ne sont pas exceptionnelles, loin s'en faut. Dans [12], les auteurs revendiquent une performance de l'ordre du Giga–Flop pour une transformée de Fourier rapide de leur cru sur la Connection Machine. La performance annoncée de la Transformée de Fourier rapide de la librairie CMSSL [23] se situe quant à elle aux alentours de 300 Mflops. Pour obtenir ces performances, il faut remplir plusieurs conditions :

- Tout d'abord, les tests mesurant les performances des transformées de Fourier se placent *toujours* dans les configurations les plus favorables. L'ordinateur frontal et la machine sont complètement monopolisés durant la mesure.
- Ensuite, la performance est évaluée pour un parallélisme poussé au maximum, et les mesures les plus favorables sont obtenues en faisant un grand nombre de transformées de Fourier de longueur relativement modeste.
- De plus, toujours dans ces tests, on ne demande pas à la machine de réordonner les résultats.
- Enfin, on appelle la transformée de Fourier avec la meilleure liste d'appel possible du point de vue de la performance.

Dans le cas présent, il s'agit de savoir quels choix doivent être faits pour assurer la meilleure performance au programme complet, et non pas seulement à la transformée de Fourier. Malheureusement, comme on l'a signalé plus haut, l'obligation que l'on a de calculer des dérivées en x *et* en y nous empêche de choisir un rangement non symétrique : il faudrait en effet changer le rangement avant et après les dérivations dans l'une des directions spatiales. Or, ce genre d'opérations est très coûteux en temps de communication car il fait appel au mécanisme de communications général. Le rangement `(1:SEND,1:SEND)` qui permet d'éviter les réagencements des tableaux semble le mieux adapté dans notre cas. Il est en fait pratiquement équivalent au `(1:NEWS,1:NEWS)` lorsque l'on demande à la transformée de Fourier rapide de délivrer son résultat en ordre binaire normal.

6.1.2 Ordre binaire inverse

L'algorithme de transformée de Fourier que l'on a examiné présente (on l'a souligné) la particularité de modifier l'adressage des éléments du vecteur résultat par rapport au vecteur données. La performance maximum de la transformée de Fourier n'est atteinte que si l'on ne lui demande pas explicitement de réordonner les éléments du tableau résultat. Or le calcul des coefficients de la dérivée par un algorithme de type "scanning" requiert un rangement normal des coefficients de Tchebychev de la fonction. On a donc implanté une version des relations de récurrence sous forme de multiplication de matrices; la matrice de "récurrence" permet, à partir des coefficients Tchebychev de la fonction, rangés en ordre binaire inverse, de calculer les coefficients de la dérivée, et de les renvoyer dans l'ordre binaire inverse; la deuxième transformée de Fourier peut alors opérer sans réagencer les éléments, et renvoyer les valeurs de la dérivée aux points de collocation rangées dans l'ordre habituel. Le cas test que l'on se propose de réaliser est bien loin des conditions optimales de fonctionnement de la machine.

On a donc comparé la performance de cette façon de procéder avec une dérivation en $\log_2(N)$, où les deux transformées de Fourier rendent leurs résultats dans l'ordre des données. Il est évidemment clair que pour des nombres de points suffisamment grands, la dérivation classique serait plus performante (parce que le fait de calculer les relations de récurrence par multiplication de matrices fait passer la complexité de la dérivation de $O(\log_2 N)$ à $O(N)$) mais il est possible que pour des nombres de points relativement petits (129 ou 257) le gain sur le réagencement des tableaux soit plus important que la perte due à la différence de complexité des deux manières de procéder.

On peut voir, table (2), que même pour des nombres de points petits, la troisième méthode est bien plus performante que celle qui autorise les transformées de Fourier à travailler en ordre binaire inverse. De plus, dans le premier cas, les multiplications de matrices se font sur des tableaux 129×256, alors que dans le second, elles ne concernent que des matrices 129×129, ce qui explique la plus grande rapidité de la seconde méthode.

6.1.3 Dérivation par paquets

Comme l'on peut s'en rendre compte dans l'exposé de l'algorithme (3.3), on a tendance, naturellement, à vouloir calculer les dérivées (dans une même

Type de dérivation	Temps CPU (s)
TFR Ordre binaire inverse et multiplication de matrices	3.2165
TFR Ordre binaire normal et multiplication de matrices	2.42554
TFR Ordre binaire normal et "scanning" pour les relations de récurrence	1.20428

Tableau 2 : Performances comparées des diverses méthodes pour calculer une dérivée (le nombre de points est de 128×128, soit une longueur de Transformée de Fourier de 256, 5 dérivations).

direction) en séquence — d'abord la masse volumique, puis la vitesse horizontale, etc... — ce qui est parfaitement normal pour un ordinateur scalaire ou même vectoriel.Une analyse plus poussée permet de se rendre compte que cette façon de procéder , si elle a le mérite de la lisibilité algorithmique, n'est pas efficace. Lorsque l'on désire calculer les transformées de Fourier de plusieurs vecteurs de données, on peut, pourvu que leur longueur soit identique et pas démesurée, appliquer simultanément l'algorithme "butterfly" à chacun des vecteurs. Dans ces conditions, il est manifeste que l'algorithme n'est pas seulement parallèle dans la direction de la transformée de Fourier, mais aussi dans l'autre. On peut alors tirer parti de cette remarque et regrouper dans un grand tableau toutes les quantités à dériver, comme si on l'avait augmenté la valeur du nombre de points dans la direction orthogonale à la direction de dérivation. Lorsque le nombre de transformées de Fourier est suffisamment élevé, les processeurs physiques simulent automatiquement plusieurs processeurs virtuels. Les accélérateurs de calcul flottants ont été spécialement concus pour alimenter leur unité de calcul de manière continue si possible (le pipe–line scalaire) : ils chargent l'opération à effectuer, puis l'effectuent tant qu'il y a des opérandes, sans recharger chaque fois l'opération.

Le tableau (3) compare les temps nécessaires pour calculer une dérivée, obtenus avec ces deux méthodes, pour plusieurs couples de nombres de points,

Type de dérivation	Temps CPU (s)
5 dérivées séparées	1.20428
1 dérivée groupée	0.370799

Tableau 3 : Performances comparées de la dérivation par paquets et de la dérivation séquentielle (le nombre de points est de 128×128, soit une longueur de Transformée de Fourier de 256), la méthode est la troisième (2).

et pour cinq dérivations. Le nombre cinq a été retenu car c'est le plus grand nombre de dérivations simultanées que l'on fait dans un module (ici le calcul des contributions dûes à la partie Euler des équations).

On constate que l'on a multiplié par un facteur proche de quatre l'efficacité du calcul des dérivées.

6.2 Expérience Numérique

On désire maintenant évaluer la performance du code implanté sur la Connection Machine, et la comparer avec celle d'un code similaire fonctionnant sur Convex C-2, CRAY-YMP.

Nombre de points	65×65	129×129	257×257
Convex C-2	21.48	175.98	613.71
Cray YMP	4.53	13.64	54.29
Connection Machine	59.29	283.55	834.24

Tableau 4 : Temps CPU (en secondes) nécessaire au calcul de 10 pas de temps pour les trois machines, Convex C-2, Cray YMP et Connection-Machine.

Le test porte sur la mesure du temps nécessaire à effectuer dix pas de temps sur chacune de ces machines, attendu que les deux codes effectuent rigoureusement les mêmes calculs. Les mesures ont été faites avec plusieurs valeurs des nombres de points et elles sont reportées dans le tableau (4). On constate que le Cray YMP est nettement plus rapide que les deux autres

machines. En ce qui concerne le Convex et La Connection Machine, l'avantage initial du Convex s'écroule dès que la taille du problème augmente.

Les résultats de ces comparaisons sont nettement en défaveur de la Connection Machine. Plusieurs explications permettent de mieux comprendre les raisons :

Le calcul d'une Transformée de Fourier rapide se fait sur le CRAY YMP dix fois plus vite que sur la Connection Machine, pour la répartition mémoire et les tailles de tableaux retenus.

Pour le rangement mémoire utilisé dans le programme de calcul, on a bien un facteur dix approximativement entre les deux machines. Comme il faut savoir que le programme utilise entre 60 et 80% du temps de calcul à calculer des Transformées de Fourier, on peut davantage comprendre le résultat du tableau (4).

Taille × nombre de TFR	1024 × 128	512 × 256	256 × 512
CRAY-YMP	0.014	0.013	0.012
CM-2	0.145	0.126	0.137

Tableau 5 : Temps d'exécution mesurés (en secondes, moyenne sur 100) de la transformée de Fourier sur la Connection Machine et sur le Cray (on s'est placé dans les conditions de calcul, et non pas dans les conditions optimales pour chaque machine).

La parallélisation n'est pas totale; pour la première des tailles du problème testées, aucune des affectations en parallèle et aucun des algorithmes comme le "parallel prefix" ou les algorithmes de réduction ne peuvent travailler à plein rendement, attendu qu'il y a des processeurs inactifs sur une CM-2 avec 8192 processeurs physiques (3967, presque la moitié); les transformées de Fourier, dont la longueur est de deux fois la dimension de l'un des axes, tirent mieux parti des processeurs disponibles, puisqu'alors aucun n'est inactif. Néanmoins, les tailles de problème sont vraisemblablement trop faibles encore pour amener la CM-2 à sa performance maximum.

L'optimisation du programme n'est sans doute pas assez poussée. Il convient de remarquer cependant que les versions CRAY et CM-2 sont à un niveau d'optimisation comparable.

Une dernière remarque s'impose enfin. Il n'est pas possible, lorsqu'on utilise une méthode spectrale pour simuler un écoulement instationnaire, de travailler avec des nombres de points trop élevés. En effet, on a un critère de stabilité pour la méthode explicite utilisée ici qui donne un pas de temps maximum admissible de l'ordre de $1/N^4$, où N est le nombre de points (en x ou en y). Il est donc clair que plus le nombre de points augmente, plus la simulation requiert un nombre important de pas de temps : le passage de $N = 64$ à $N = 128$ divise le pas de temps approximativement par 16 (approximativement seulement car le critère de pas de temps est complexe et fait aussi intervenir des $1/N^2$). Or on sait que la performance d'une machine parallèle croit avec le nombre de points. Dans le cas d'une machine massivement parallèle, il faudrait atteindre une taille de calcul colossale (512^2 ou 1024^2) pour que les possibilités de la machine soient pleinement utilisées.

6.3 Simulation numérique d'une couche de mélange se développant temporellement

Le cas test est celui présenté dans la section (2).

Le domaine de calcul est le carré $[0, 2\lambda_a] \times [-\lambda_a, \lambda_a]$, où $\lambda_a = \tilde{\lambda_a}\delta_i$ est la longueur d'onde la plus instable prédite par la théorie linéaire.

Nous supposerons en outre que le rapport des chaleurs spécifiques γ est égal à 1.4 et que le nombre de Prandtl est lui égal à 0.7. Nous prendrons de plus $S_1 = 110K$. Enfin, le calcul présenté concernent le cas $Re = 1000$ et $M_\infty = 0.3$. La longueur d'onde la plus instable vaut dans ce cas $\lambda_a = 7.53$.

On s'aperçoit que les tourbillons se sont formés à partir de l'écoulement de base et de la perturbation initiale. Les champs de pression et de masse volumique montrent des variations près des bords inférieur et supérieur du domaine, qui sont dues à la réflexion sur le bord du domaine des ondes acoustiques engendrées par la couche, comme expliqué dans [17]. Les champs sont réguliers et exempts d'oscillations, mais les résultats ne sont pas significatifs d'un point de vue purement physique. Il s'agissait seulement de montrer que le code parallèle est capable de calculer correctement un couche de mélange compressible.

7 Conclusion

Nous avons étudié ici le comportement de la méthode spectrale appliquée à la simulation d'écoulements compressibles sur une machine massivement parallèle, la CM–2. Les résultats présentés montrent à l'évidence que ce comportement est décevant. Certes des optimisations sont encore possibles et le rapport des temps de calcul avec le CRAY–YMP pourraient probablement être ramenés de 15 actuellement à 10 par exemple, mais cela resterait néanmoins décevant en regard des efforts fournis pour porter le programme sur l'une et l'autre des machines.

Les raisons d'un tel comportement ont été relevées et expliquées dans la dernière partie de ce rapport. Rappelons pour mémoire qu'en ce qui concerne le calcul des transformées de Fourier, nous avons utilisé la routine disponible dans la librairie fournie, qui a une performance maximale de 300 Mflops. Des algorithmes beaucoup plus performants ont été construits auxquels nous n'avons pas pu accéder. Nous avons montré aussi qu'il nous était impossible de nous placer dans les conditions optimales d'utilisation de la routine de transformée de Fourier pour prétendre approcher ces performances, à cause de la nature même du problème que l'on veut résoudre. En effet, on ne peut fixer arbitrairement la longueur et le nombre de transformées de Fourier; elles se déduisent des dimensions du maillage utilisé. Or ces dimensions ne peuvent être choisies trop grandes, au risque de diminuer par trop la valeur du pas de temps nécessaire, et donc d'exiger un nombre faramineux de pas de temps pour parvenir à un temps donné. Est–ce à dire que la méthode spectrale est intrinsèquement non parallèle ? L'analyse des dépendances de l'algorithme montre que chaque étape du calcul peut s'effectuer en parallèle, mais les expériences numériques ont mis en évidence les limites de parallélisme. C'est l'aspect massif de ce parallélisme de la Connection Machine qui est à l'origine des résultats obtenus et non pas le synchronisme, l'aspect SIMD ou la topologie de type hypercube par exemple.

En conclusion, nous pensons que le comportement de l'algorithme décrit ici pourra être très différent sur des machines à grain moyen avec des processeurs locaux beaucoup plus rapides que les processeurs bit–à–bit de l'actuelle CM2.

A Expression de la matrice dérivée première

Les cœfficients de la matrice de dérivation $[d_{jk}^{(1)}]_{jk}$ s'écrivent :

$$\begin{cases} d_{00}^{(1)} = -d_{NN}^{(1)} = \dfrac{2N^2+1}{6} & \\ d_{jj}^{(1)} = -\dfrac{x_j}{2(1-x_j^2)} & j \neq 0, N \\ d_{jk}^{(1)} = \dfrac{\overline{c}_k}{\overline{c}_j} \dfrac{(-1)^{j+k}}{(x_j - x_k)} & j \neq k \end{cases} \tag{24}$$

Références

[1] Betchov (R.) et Szewczyk (G.). – Stability of a shear layer between parallel streams. *Phys. Fluids*, vol. 170, 1963, pp. 499–525.

[2] Blumen (W.). – Shear layer instability of an inviscid compressible fluid. *J. Fluid Mech.*, vol. 40, 1970, pp. 769–781.

[3] Blumen (W.), Drazin (P. G.) et Billings (D. F.). – Shear layer instability of an inviscid compressible fluid, Part II. *J. Fluid Mech.*, vol. 71, 1975, pp. 305–316.

[4] Bogdanoff (D. W.). – Compressibility effects in turbulent shear layers. *AIAA J.*, vol. 21, 1983, pp. 926–927.

[5] Canuto (C.), Hussaini (M. Y.), Quarteroni (A.) et Zang (T. A.). – *Spectral Methods in Fluid Dynamics.* – Springer–Verlag, 1988, *Springer Series in Computational Physics.*

[6] Desfontaines (C.). – *Produit parallèle de matrices sur la Connection Machine.* – Mémoire, ESSI, option CSI, 1991.

[7] Djordjevic (V. D.) et Redekopp (L. G.). – Linear stability analysis of nonhomentropic, inviscid compressible flows. *Phys. Fluids*, vol. 31, 1988, pp. 3239–3245.

[8] Gama (S.), Frisch (U.) et Scholl (H.). – The 2–D Navier–Stokes equations with a large scale instability of the Kuramoto–Sivashinsky type :

numerical exploration on the Connection-Machine. *Submitted, J. Sci. Comp.*

[9] Guillard (H.), Malé (J. M.) et Peyret (R.). – Numerical simulation of compressible mixing layer using adaptive spectral methods. *In : Proceedings of th* 4^{th} *International Symposium on Computational Fluid Dynamics.* pp. 449–454. – University of California, Davis. 1991.

[10] Guillard (H.), Malé (J. M.) et Peyret (R.). – Adaptive spectral methods with application to mixing layer computations. *J. Comput. Phys.*, vol. 102, n° 1, 1992, pp. 114–127.

[11] Johnsson (S. L.), Ho (C. H.), Jacquemin (M.) et Ruttenberg (A.). – Computing fast Fourier transforms on boolean cubes and related networks. *In : Advanced Algorithms and Architecture for signal processing II.*, éd. par Krause. pp. 223–231. – Int. Soc. Opt. Eng. San Diego CA, USA, 18–19 Aug. 1987.

[12] Johnsson (S. L.), Ho (C. H.), Jacquemin (M.) et Ruttenberg (A.). – Systolic FFT algorithms on boolean cube networks. *In : Proceedings of the International Conference on Systolic Arrays.* pp. 151–162. – Washington DC USA. San Diego, CA, USA 25–27 May 1988.

[13] Johnsson (S. L.), Krawitz (R. L.), Frye (R.) et MacDonald (D.). – A radix–2 FFT on the Connection Machine. *In : Proceedings of Supercomputing'89.* – New–York, NY, USA. Reno, NV, USA, 13–17 Nov. 1989.

[14] Kamin III (R. A.) et Adams III (G. B.). – *Fast Fourier Algorithm design and Tradeoffs on the CM–2.* – Technical Report n° 88-18, RIACS, 1988.

[15] Lantéri (S.), Farhat (C.) et Fézoui (L.). – *Structured Compressible Flow Computation on the Connection Machine.* – Rapport n° 1322, INRIA, 1990.

[16] Levit (C.). – *Grid Communication on the Connection Machine, Analysis, Performance and Improvements.* – 1988, *Scientific Applications of the Connection Machine.*

[17] Malé (J. M.). – *Calcul d'écoulements visqueux en Méthodes Spectrales Auto-Adaptatives. Application aux couches de Mélange.* – Phd thesis, Université de NICE, 1992. In preparation.

[18] Michalke (A.). – On the inviscid instability of the hyperbolic-tangent velocity profile. *J. Fluid Mech.*, vol. 19, 1964, pp. 543–556.

[19] Papamoschou (D.) et Roshko (A.). – The compressible turbulent shear layer : An experimental study. *J. Fluid Mech.*, vol. 197, 1988, pp. 453–477.

[20] Sandham (N. D.) et Reynolds (W.). – The compressible mixing layer : Linear theory and direct simulation. *AIAA J.*, vol. 28, 1989, pp. 618–624.

[21] Thinking Machines Corporation. – *PARallel Instruction Set*, 1989. Connection Machine documentation set.

[22] Thinking Machines Corporation. – *C* Programming Guide*, 1990. Connection Machine documentation set.

[23] Thinking Machines Corporation. – *Connection Machine System Scientific Library*, 1991. Connection Machine documentation set.

[24] Thinking Machines Corporation. – *Getting started in CM-Fortran*, 1991. Connection Machine documentation set.

[25] Wengle (H.) et Seinfeld (J.H.). – Pseudo spectral solution of atmospheric diffusion problems. *J. Comput. Phys.*, vol. 26, 1978, pp. 87–106.

[26] Williamson (J.H.). – Low-storage Runge–Kutta schemes. *J. Comput. Phys.*, vol. 35, 1980, pp. 48–56.

Table des matières

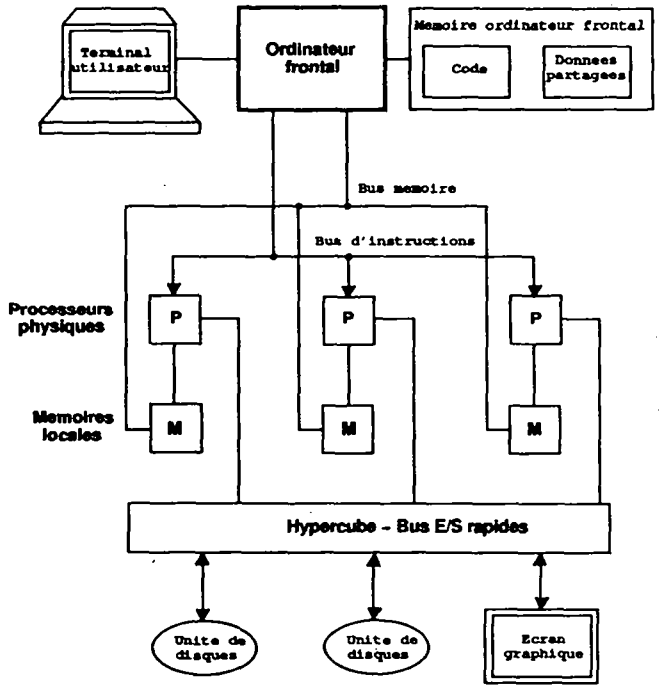


Figure 2 : Organisation de la Connection Machine CM-2

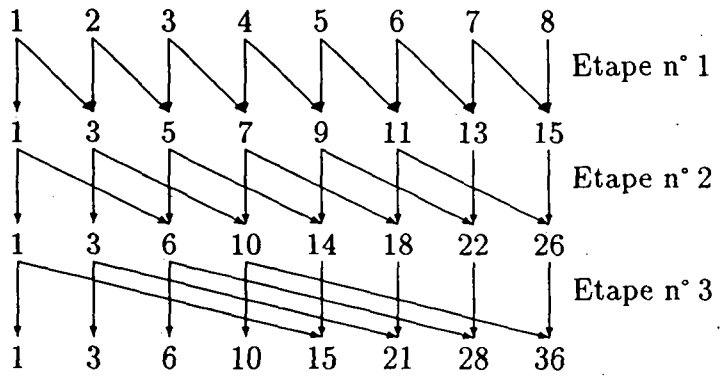


Figure 3 : Algorithme de calcul des relations de récurrence

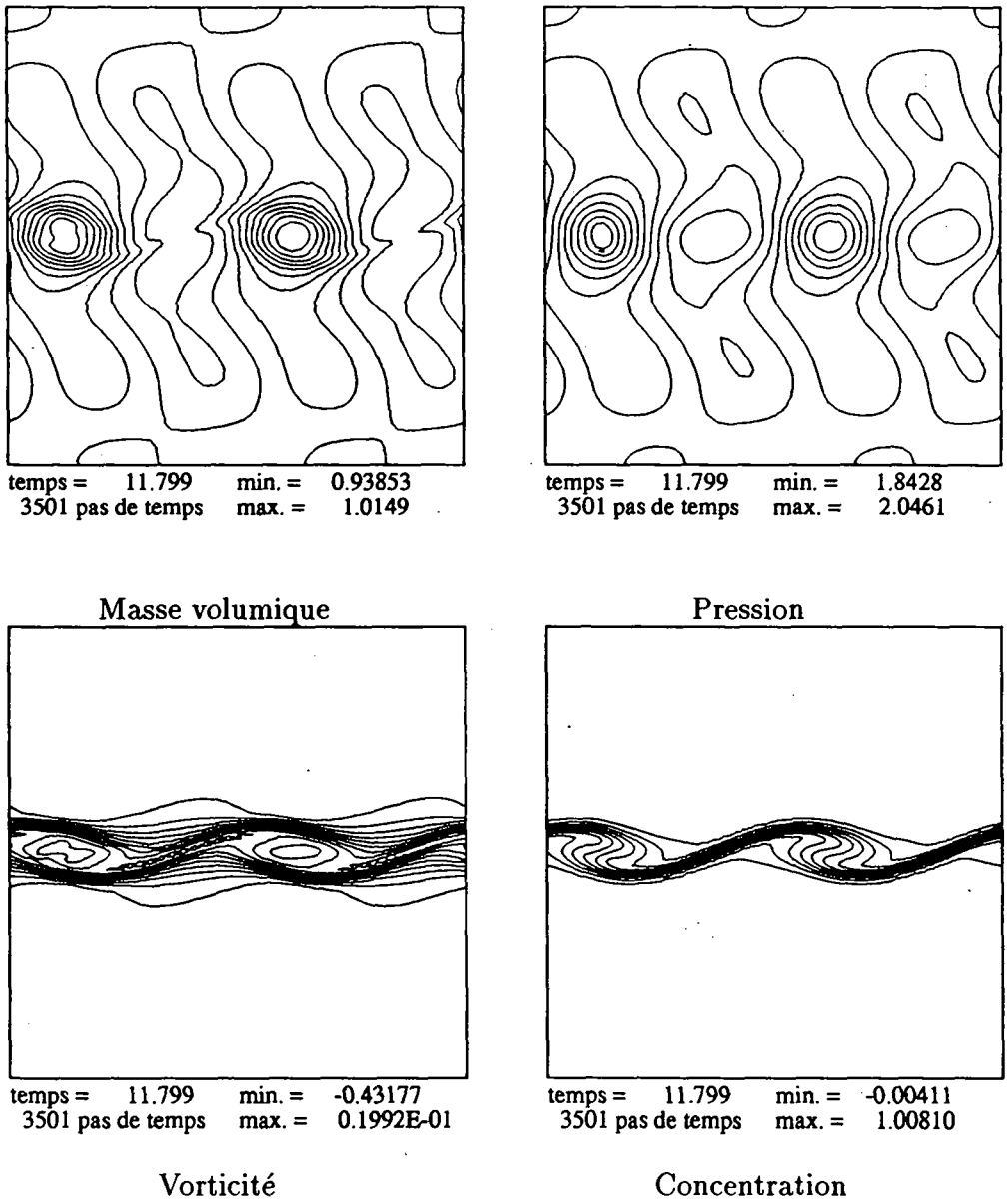


Figure 4 : Panorama de l'écoulement à $t = 11.8$.

Unité de Recherche INRIA Sophia Antipolis
2004, route des Lucioles - B.P. 93 - 06902 SOPHIA ANTIPOLIS Cedex (France)

Unité de Recherche INRIA Lorraine Technopôle de Nancy-Brabois - Campus Scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 VILLERS LES NANCY Cedex (France)
Unité de Recherche INRIA Rennes IRISA, Campus Universitaire de Beaulieu 35042 RENNES Cedex (France)
Unité de Recherche INRIA Rhône-Alpes 46, avenue Félix Viallet - 38031 GRENOBLE Cedex (France)
Unité de Recherche INRIA Rocquencourt Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 LE CHESNAY Cedex (France)

EDITEUR
INRIA - Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 LE CHESNAY Cedex (France)

ISSN 0249 - 6399